

### 5.1) MIPS

#Aufgabe 5.1)  
#Georg Kuschik

#ACHTUNG : Da laut Forum davon ausgegangen werden soll, dass der Eingabewert,  
# falls er denn kleiner gleich 10 Stellen besitzt,  
# mit maximal 32Bit darstellbar ist, wird nicht noch extra ueberprueft  
# ob der Eingabewert zwischen 4294967296 und 9999999999 liegt  
#  
# Ebenso werden eventuell fuehrende Nullen nicht entfernt, sonder als  
# Dezimalstelle mitgezaehlt

```
.data
str_txt1: .asciiz  "\nEingabe : "
str_txt2: .asciiz  "\nFehler! - Keine gueltige Eingabe !"
str_txt3: .asciiz  "\nProgrammende - letztes Rechenergebnis : "
str_txt4: .asciiz  "Noch keine Zahl auf den Stack geschrieben !"
str_txt5: .asciiz  "\nKeine 2 Operanden fuer die Operation auf dem Stack vorhanden !"
str_txt6: .asciiz  "\nKann Division durch Null nicht ausfuehren !"
str_txt7: .asciiz  "\nErgebnis der Operation : "
```

```
str_input: .space 64
```

```
.text
```

```
main:
##### Initialisierung der globalen Variablen #####
li      $s1,0          #in $s1 steht die Anzahl der auf dem Stack befindlichen Zahlen
```

```
loop_input:
li      $v0,04          #print_string "Eingabe : "
la      $a0,str_txt1
syscall
```

```
##### Zeichenkette einlesen #####
li      $v0,08          #read_string - EingabeString in str_input einlesen
la      $a0,str_input   #Adresse des EingabeStrings laden
li      $a1,64          #maximale Laenge des EingabeStrings
syscall          #schliesslich die Funktion (read_string) aufrufen
```

```
##### Zeichenkette auswerten #####
```

```
##### Prüfen, ob 'q' eingegeben wurde #####
```

```
li      $t1,0
li      $s0,1
lb      $t1,str_input($s0) #pruefen, ob die Zeichenkette nur ein Zeichen enthaelt (2.Byte->Wert=10)
bne     $t1,10,check_number #wenn nicht, dann nicht als Operationssymbol erkennen
```

```
lb      $t1,str_input($zero) #1. Zeichen der Zeichenkette in $t1 laden und auswerten
bne     $t1,q,check_symbol #wenn das Zeichen ungleich 'q' ist, dann prüfen, ob
                        #die Zeichenkette ein Operationssymbol darstellt
```

```
                        #wenn Zeichen ='q' :
li      $v0,04          #print_string "\nProgrammende - letztes Rechenergebnis : "
la      $a0,str_txt3
syscall
```

```
bne     $s1,$zero,stack_print #befinden sich auf dem Stack keine eingegebenen Zahlen
                        #oder Rechenergebnisse so wird eine Meldung angezeigt,
                        #ansonsten wird der letzte Wert vom Stack geholt und ausgegeben
```

```

li      $v0,04                      #print_string "Noch keine Zahl auf den Stack geschrieben !"
la      $a0,str_txt4
syscall
j       exit_prog                  #und zum Programm-Ende springen

#####Falls sich auf dem Stack Werte befand, so wird der "oberste" hier ausgegeben
stack_print:
li      $v0,01                      #print_int
lw      $a0,4($sp)                  #letzten Wert vom Stack holen
addi    $sp,4                      #Stackpointer aktualisieren
syscall
addi    $s1,-1                      #Zahlen-Counter dekrementieren
j       exit_prog                  #und zum Programm-Ende springen


check_symbol:
##### Prüfen, ob ein Operationssymbol +,-,*,/ eingegeben wurde #####
lb      $t1,str_input($zero)        #1. Zeichen der Zeichenkette in $t1 laden und auswerten

lb      $t1,str_input($zero)        #1. Zeichen der Zeichenkette in $t1 laden und auswerten

beq     $t1,'+',addition             # + -> springe zur Addition
beq     $t1,'-',subtraction          # - -> springe zur Subtraktion
beq     $t1,'*',multiplication       # * -> springe zur Multiplikation
beq     $t1,'/',division             # / -> springe zur Division

j       check_number               #wurde kein gueltiges Operationssymbol eingegeben, pruefen,
                                   #ob die Zeichenkette eine Zahl ist


addition:
blt     $s1,2,no_ops                #Prüfen, ob mindestens zwei Operanden auf dem Stack liegen
lw      $t0,8($sp)                  #1.Operanden vom Stack holen holen
lw      $t1,4($sp)                  #2.Operanden vom Stack holen holen
addi    $sp,8                      #Stackpointer aktualisieren
addi    $s1,-2                      #Zahlen-Counter um die zwei soeben gelesenen Werte verringern
add     $t0,$t0,$t1                 #Addition ueber den zwei Operanden ausfuehren
j       op_success                  #Ergebnis berechnet

subtraction:
blt     $s1,2,no_ops                #Prüfen, ob mindestens zwei Operanden auf dem Stack liegen
lw      $t0,8($sp)                  #1.Operanden vom Stack holen holen
lw      $t1,4($sp)                  #2.Operanden vom Stack holen holen
addi    $sp,8                      #Stackpointer aktualisieren
addi    $s1,-2                      #Zahlen-Counter um die zwei soeben gelesenen Werte verringern
sub     $t0,$t0,$t1                 #Subtraktion ueber den zwei Operanden ausfuehren
j       op_success                  #Ergebnis berechnet

multiplication:
blt     $s1,2,no_ops                #Prüfen, ob mindestens zwei Operanden auf dem Stack liegen
lw      $t0,8($sp)                  #1.Operanden vom Stack holen holen
lw      $t1,4($sp)                  #2.Operanden vom Stack holen holen
addi    $sp,8                      #Stackpointer aktualisieren
addi    $s1,-2                      #Zahlen-Counter um die zwei soeben gelesenen Werte verringern
mul     $t0,$t0,$t1                 #Multiplikation ueber den zwei Operanden ausfuehren
j       op_success                  #Ergebnis berechnet

division:
blt     $s1,2,no_ops                #Prüfen, ob mindestens zwei Operanden auf dem Stack liegen
lw      $t0,8($sp)                  #1.Operanden vom Stack holen holen
lw      $t1,4($sp)                  #2.Operanden vom Stack holen holen
addi    $sp,8                      #Stackpointer aktualisieren
addi    $s1,-2                      #Zahlen-Counter um die zwei soeben gelesenen Werte verringern

beqz    $t1,divnull                 #Division durch Null abfangen
div     $t0,$t0,$t1                 #wenn Divisor ungleich Null, dann Ergebnis berechnen
j       op_success                  #Ergebnis berechnet

divnull:
#Division durch Null anzeigen
li      $v0,04                      #print_string "\nKann Division durch Null nicht ausfuehren !"
la      $a0,str_txt6
syscall
j       loop_input                  #bei Division durch Null gleich wieder neue Zeichenkette erfragen

```

```

op_success:
    ###bei erfolgreich ausgefuehrter Operation das Ergebnis (liegt in $t0) auf den Stack schreiben
    addi    $sp,-4                #Stackpointer dekrementieren
    sw      $t0,4($sp)            #Ergebnis auf den Stack schreiben
    add     $s1,$s1,1             #Zahlen-Counter inkrementieren

    ###und auch noch auf der Konsole ausgeben
    li      $v0,04                #print_string "\nErgebnis der Operation : "
    la      $a0,str_txt7
    syscall
    li      $v0,01                #print_int
    add     $a0,$t0,0             #Ergebnis der letzten Rechnung der Funktion als Argument uebergeben
    syscall
    j       loop_input            #und wieder neue Zeichenkette erfragen

no_ops:
    li      $v0,04                #print_string "\nKeine 2 Operanden fuer die Operation auf dem Stack vorhanden !"
    la      $a0,str_txt5
    syscall
    j       loop_input            #und wieder neue Zeichenkette erfragen

```

```

check_number:
    ##### Prüfen, ob die Eingabe eine gueltige, vorzeichenlose Dezimalzahl ist #####
    li      $s0,0                #$s0 zaehlt die Anzahl der Ziffern

input_evaluation:
    lb      $t1,str_input($s0)    #Zeichen des eingegebenen Strings einzeln in $t1 laden und auswerten

    beq     $t1,10,input_end      #Ende des Eingabestrings erreicht?
    blt     $t1,'0',input_error   #wenn nicht : ist das Zeichen eine Ziffer ('0'-'9')?
    bgt     $t1,'9',input_error

    add     $s0,$s0,1             #Zeichen ist eine Ziffer -> Anzahl der Ziffern inkrementieren
                                    #und den Integer-Wert der Ziffer auf den Stack schreiben

    addi    $sp,-1                #Stackpointer dekrementieren
    sub     $t1,$t1,'0'           #das gelesene Zeichen in entsprechende Ziffer umrechnen
    sb      $t1,1($sp)            #und die Ziffer auf den Stack schreiben

    j       input_evaluation      #da das Ende des Eingabestrings noch nicht erreicht ist,
                                    #das naechste Zeichen auswerten

```

```

input_end:
    beq     $s0,$zero,input_error #hat die Zahl 0 Stellen ? -> dann Fehler
    bgt     $s0,10,input_error    #hat die Zahl > 10 Stellen ? -> dann Fehler

    ##### EingabeString wurde jetzt akzeptiert -> Binaerzahl berechnen
    ##### zur Erinnerung : Anzahl der Ziffern der Dezimalzahl steht in $s0 und muss an dieser Stelle >0 sein
    ##### und die Ziffern stehen in umgekehrter Reihenfolge auf dem Stack

    li      $t0,0                #das Gesamtergebnis wird zunaechst in $t0 zwischengespeichert
    li      $t1,1                #in $t1 stehen die gerade benoetigten 10er-Potenzen

dec2bin_loop:
    li      $t2,0                #$t2 fuer Aufnahme eines Bytes vorbereiten
    lb      $t2,1($sp)            #Ziffer vom Stack holen
    addi    $sp,1                #Stackpointer aktualisieren

    mul     $t2,$t2,$t1           #Ziffern-Wertigkeit entsprechend der Dezimal-Stelle berechnen
    add     $t0,$t0,$t2           #und zum Gesamtergebnis hinzuaddieren

    mul     $t1,$t1,10            #naechste 10er-Potenz berechnen
    sub     $s0,$s0,1            #Anzahl der noch abzuarbeitenden Stellen dekrementieren
    bgt     $s0,$zero,dec2bin_loop #und wenn diese >0 ist, mit dem Verfahren fortfahren

    ##### Die bisher von dem Programm-Teil "check_number" auf den Stack geschriebenen Ziffern wieder entfernen
    add     $sp,$sp,$s0           #Stackpointer wieder um die Anzahl der auf den Stack geschriebenen
                                    #Bytes erhoehen

    #Alle Ziffern wurden abgearbeitet , die Zahl ist erkannt und steht nun in $t0 -> Zahl auf den Stack schreiben
    addi    $sp,-4                #Stackpointer dekrementieren
    sw      $t0,4($sp)            #Zahl auf den Stack schreiben
    add     $s1,$s1,1            #Zahlen-Counter inkrementieren

```

```
j      loop_input      #und wieder neue Zeichenkette erfragen
```

##### Hier ist das Sprungziel des Programms fuer eine ungueltige Eingabe

input\_error:

```
li      $v0,04          #print_string "\nFehler! - Keine gueltige Eingabe !"
la      $a0,str_txt2
syscall
```

```
##### Die bisher von dem Programm-Teil "check_number" auf den Stack geschriebenen Ziffern wieder entfernen
add     $sp,$sp,$s0      #Stackpointer wieder um die Anzahl der auf den Stack geschriebenen
                        #Bytes erhoehen
```

```
j      loop_input      #und wieder neue Zeichenkette erfragen
```

exit\_prog:

```
#      ##### auf Tastendruck zum Beenden warten #####
#      li      $v0,05      #read_int
#      syscall
```

```
li      $v0,10          #exit
syscall
```

## 5.2 MIPS-Pipelining

- Die dem Branch folgenden Befehle dürfen im Falle einer Verzweigung keine Wirkung haben.
- D.h. alle gesetzten Schreibsignale dieser Befehle sind von der gesuchten Logik zu ignorieren.
- Hierbei handelt es sich um Befehle aus den 3 Phasen Execute, Decode und Fetch.
  - In der Execute-Phase wird gerade der Befehl verarbeitet, der nach dem Branch Befehl stand.
  - Es folgt der Befehl aus der Decode-Phase,
  - sowie der Befehl aus der Fetch-Phase der gerade erst geladen wird.

In der Fetch-Phase kann man mit Hilfe eines Multiplexers, welcher über die Signalleitung PCSrc gesteuert wird zwischen NOP und dem eigentlichen Befehl wählen lassen.

Ist PCSrc=0 (d.h. es wird nicht gesprungen), dann leitet der Multiplexer den geladenen Befehl einfach weiter.

Ist PCSrc=1 (d.h. es wird gesprungen), leitet er den NOP – Befehl weiter.

In der Execution-Phase und der Decode-Phase kann man für den Fall das gesprungen wird (PCSrc=1) die Leitungen RegWrite und MemWrite einfach auf 0 setzen.

Dies wird durch zwei AND-Verknüpfungen realisiert :

$\neg PCSrc \wedge \text{RegWrite}$                       sowie                       $\neg PCSrc \wedge \text{MemWrite}$

Hardware-Realisierung : siehe GIF-Datei